

# Structure of a server

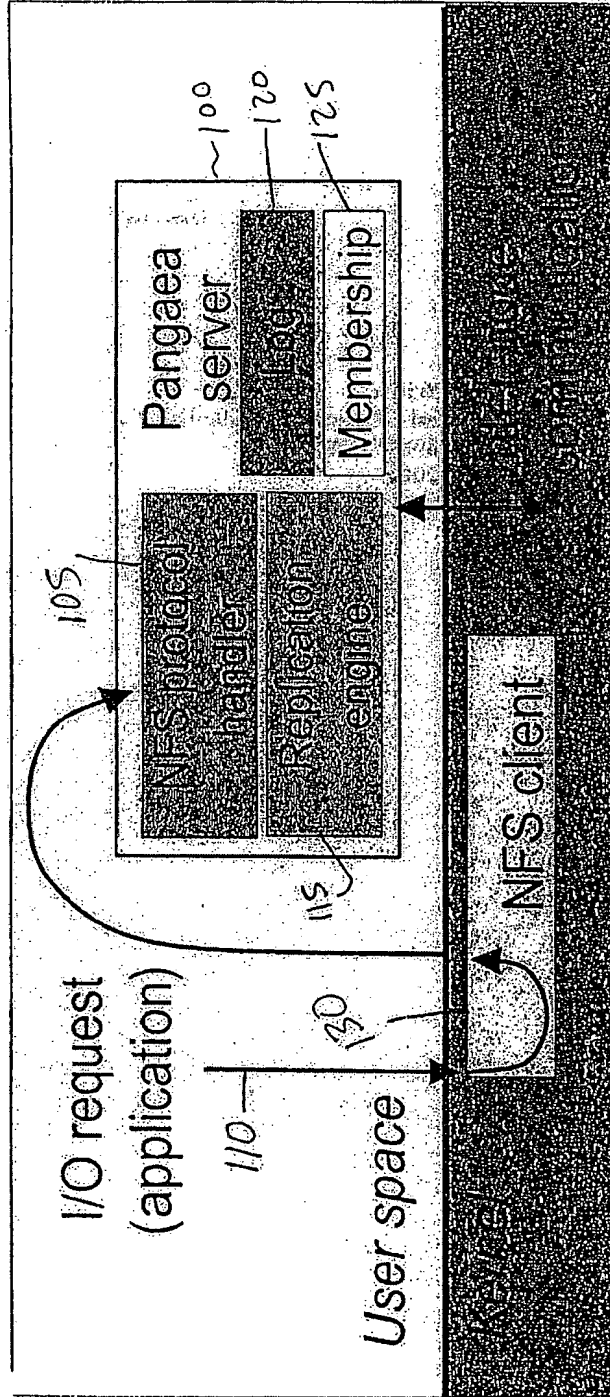


Figure 1

BEST AVAILABLE COPY

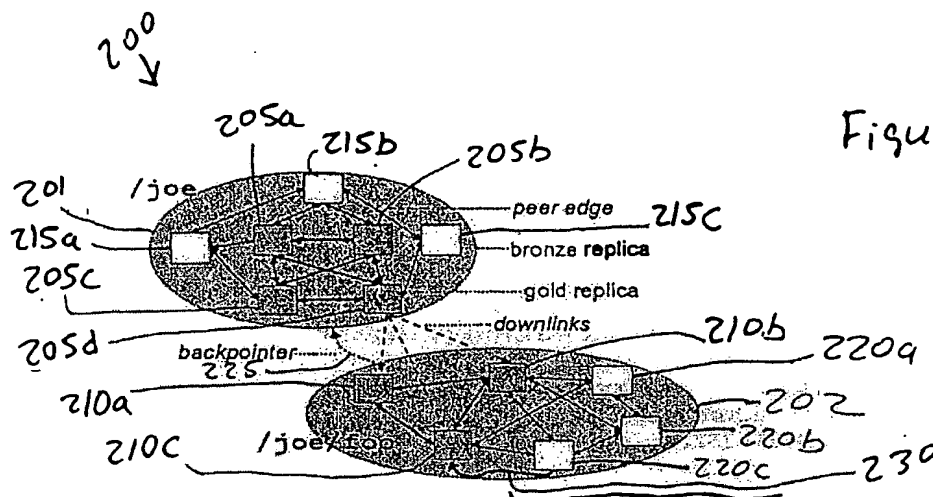


Figure 2

**Figure 2:** An example of a directory `/joe` and file `/joe/foo`. Each replica of `joe` stores three pointers to the gold replicas of `foo`. Each replica of `foo` keeps a backpointer to the parent directory. Bronze replicas are connected randomly to form strongly connected graphs. Bronze replicas also have uni-directional links to the gold replicas of the file, which are not shown here.

Key attributes of a replica

```
struct Replica
  fid: FileID           // 96 bit globally unique file ID
  ts: Timestamp         // Pair of hphysical clock, IP address
  vv: VersionVector     // Maps IP addr to Timestamp
  goldPeers: Set(NodeID) // Set of IP addresses
  peers: Set(NodeID)
  backptrs: Set(FileID, String) // Pair of hdirID, fnamei
  ...
end
struct DirEntry
  fname: String
  fid: FileID
  downlinks: Set(NodeID)
  ts: Timestamp
end
```

Figure 3

**proc** UpdateReplica

$r_2$ : **Replica** // New replica contents.

**preconditions:**

$\forall \langle pfid, fname \rangle: r_2.bptrs \bullet pfid \in \text{dom}(DISK)$   
and  $IsLive(r_2) \Rightarrow IsLive(DISK(pfid))$

**postconditions:**

$r_2.bptr \neq \{\}$  <sup>(3)</sup>

---

...

Fig. 4

- “ $\langle \text{val}_1, \dots, \text{val}_n \rangle$ ” represents a tuple of values.
- “ $\mathbb{P}$  **type**” represents a (possibly empty) set of **type**.  
“ $\mathbb{P}_1$  **type**” represents a nonempty set of **type**.  
“**Key**  $\mapsto$  **Val**” represents a one-to-many mapping from type **Key** to **Val**.
- “ $\text{dom}(F)$ ” returns the domain of function (or mapping)  $F$ , and “ $\text{ran}(F)$ ” returns the range of  $F$ . For instance,

$$\begin{aligned}\text{dom}(\{1 \mapsto 3, 2 \mapsto 8, 4 \mapsto 3\}) &= \{1, 2, 4\}, \\ \text{ran}(\{1 \mapsto 3, 2 \mapsto 8, 4 \mapsto 3\}) &= \{3, 8\}.\end{aligned}$$

- “ $X \oplus Y$ ” substitutes a part of mapping  $X$  by  $Y$ . E.g.,

$$\begin{aligned}\{1 \mapsto 3, 2 \mapsto 1\} \oplus \{1 \mapsto 5, 3 \mapsto 4\} \\ = \{1 \mapsto 5, 2 \mapsto 1, 3 \mapsto 4\}.\end{aligned}$$

- “ $X \triangleleft Y$ ” means function-domain restriction. E.g.,

$$\{2\} \triangleleft \{1 \mapsto 3, 2 \mapsto 8, 4 \mapsto 6\} = \{1 \mapsto 3, 4 \mapsto 6\}.$$

- “ $\forall \text{var. set} \bullet \text{expr}$ ” means that  $\text{expr}$  holds for  $\text{var}$  in  $S$ .  
E.g.,

$$\forall n : \{11, 13, 17\} \bullet \text{IsPrime}(n).$$

- “ $\diamond \text{expr}$ ” means that  $\text{expr}$  holds eventually.
- “ $\{\text{var. set} \bullet \text{expr}\}$ ” means set comprehension. E.g.,

$$\{x : \{1, 2, 3\} \bullet x^2\} = \{1, 4, 9\}.$$

F.5.5

```

type Replica = record
  fid(1) : FileID
  peers(2) :  $\mathbb{P}$  NodeID
  gpeers(3) :  $\mathbb{P}_1$  NodeID
  bptrs(4) :  $\mathbb{P}$  Backptr
  deadBptr(5) : Backptr
  ts(6) : Timestamp
type RegularReplica inherits Replica =
  contents(7) : Data
  Invariants:
     $\neg \text{isLive}(r) \Rightarrow \text{contents} = \{\}$ 
type DirReplica inherits Replica =
  ents(8) :  $\langle \text{FileID}, \text{String} \rangle \rightarrow \text{DEntry}$ 
  Invariants:
     $\neg \text{isLive}(r) \Rightarrow \text{ents} = \{\}$ 
type Backptr =  $\langle \text{FileID}, \text{String} \rangle$ 
type Dentry = record
  valid(9) : bool
  ts(10) : Timestamp
  gpeers(11) :  $\mathbb{P}_1$  NodeID
proc isLive(r)
  return r is the root or r.bptrs  $\neq \{\}$ 

```

Fig. 6

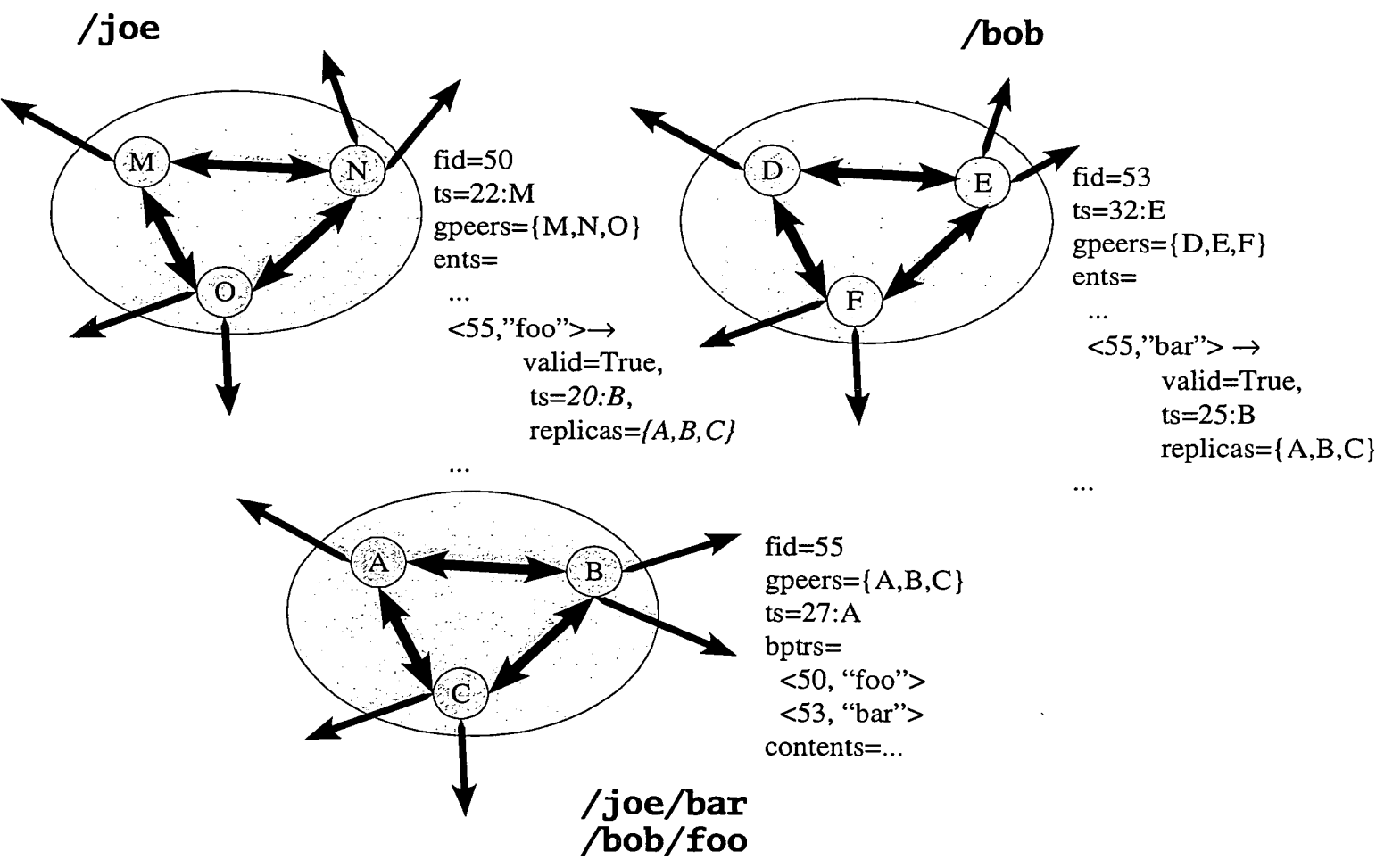


Fig. 7

***DISK*: FileID  $\mapsto$  Replica**

***CLOG*: FileID  $\mapsto \mathbb{P}_1$  NodeID**

***ULOG*: FileID  $\mapsto \mathbb{P}$  Backptr**

**Invariants::**

// Updates are only for existing replicas.

$\text{dom}(\text{CLOG}) \cup \text{dom}(\text{ULOG}) \subseteq \text{dom}(\text{DISK})$  <sup>(12)</sup>

F.3. 8



**proc Create**

**d: DirReplica** // The local replica of the parent directory.

**fname: string** // The name of the new file in *d*

**gpeers:  $\mathbb{P}_1$  NodeID** // The placement of the replicas of the file.

**preconditions:**

**IsLive(*d*)** <sup>(13)</sup>

---

*r*  $\leftarrow$  **Newreplica()**

*r.fid*  $\leftarrow$  **Newfileid()**

*r.gpeers*  $\leftarrow$  *gpeers*

*r.ts*  $\leftarrow$  **Newtimestamp()**

*r.peers*  $\leftarrow$  **{}**

*r.bptrs*  $\leftarrow$  **{⟨*d.fid*, *fname*⟩}**

*r.contents*  $\leftarrow$  **None**

**UpdateReplica(*r*)**

F.3 9

**proc Unlink**

*f*: **Replica** // The file to be unlinked.

*d*: **DirReplica** // The directory the file belongs to.

*f**name*: **string** // *f*'s name in *d*.

**preconditions:**

IsLive(*d*)

*f* is a directory  $\Rightarrow f.ents = \{\}$

$\langle d.fid, fname \rangle \in f.bptrs$

---

$f' \leftarrow \text{Deepcopy}(f)$

$f'.bptrs \leftarrow f.bptrs \setminus \{\langle d.fid, fname \rangle\}$

**if**  $f'.bptrs = \{\}$  **then**

$f'.deadBptr \leftarrow \langle d.fid, fname \rangle$

$f'.ts \leftarrow \text{Newtimestamp}()$

UpdateReplica(*f'*)

Fig. 10

**proc Hardlink**

*f*: **RegularReplica** // The replica of the file.

*d*: **DirReplica** // The directory to which *f* will be linked to.

*fname*: **string** // The filename within *d*.

**preconditions:**

IsLive(*d*)

---

*f'*  $\leftarrow$  Deepcopy(*f*)

*f'*.*bptrs*  $\leftarrow$  *f*.*bptrs*  $\cup$  { $\langle d.fid, fname \rangle$ }

*f'*.*ts*  $\leftarrow$  Newtimestamp()

UpdateReplica(*f'*)

**proc Rename**

*f*: **Replica** // The file to be moved.

*d<sub>F</sub>*: **DirReplica** // The origin dir.

*d<sub>T</sub>*: **DirReplica** // The destination dir.

*fname<sub>F</sub>*: **string** // The filename in *d<sub>F</sub>*

*fname<sub>T</sub>*: **string** // The filename in *d<sub>T</sub>*

**preconditions:**

IsLive(*d<sub>F</sub>*) and IsLive(*d<sub>T</sub>*)

$\langle d_F.fid, fname_F \rangle \in f.bptrs$

---

*f'*  $\leftarrow$  Deepcopy(*f*)

*f'.bptrs*  $\leftarrow f.bptrs \setminus \{\langle d_F.fid, fname_F \rangle\} \cup \{\langle d_T.fid, fname_T \rangle\}$

*f'.ts*  $\leftarrow$  Newtimestamp()

UpdateReplica(*f'*)

F.5. 12

**proc Write**

***f*: RegularReplica**

***newcontents*: Data**

---

$f' \leftarrow \text{Deepcopy}(f)$

$f'.\text{contents} \leftarrow \text{newcontents}$

$f'.\text{ts} \leftarrow \text{Newtimestamp}()$

$\text{UpdateReplica}(f')$

Fig. 13

```

proc UpdateReplica
   $r_2$ : R plica // New replica contents.
preconditions:
  // All parent directories are stored locally.
  // Moreover, if  $r_2$  is live, then parent must also be live.
   $\forall \langle pfid, fname \rangle: r_2.bptrs \bullet pfid \in \text{dom}(DISK)$ 
  and  $IsLive(r_2) \Rightarrow IsLive(DISK(pfid))$  (14)



---


if  $r_2.fid \notin \text{dom}(DISK)$  then
  // The replica isn't locally stored yet.
   $DISK \leftarrow DISK \cup \{ r_2.fid \mapsto r_2 \}$ 
  IssueCupdate( $r_2$ )
  return

 $r_1 \leftarrow DISK(r_2.fid)$ 

if File is regular then
  Do some application-specific stuff.
  We can potentially use version vectors here.
else
  // Union dir entries, taking ones with newer timestamps on conflict.
  for  $(key \mapsto e) \in r_2.ents$ 
    if  $key \notin \text{dom}(r_1.ents)$  or  $r_1.ents(key).ts < e.ts$ 
       $r_1.ents \leftarrow r_1.ents \oplus \{ key \mapsto e \}$ 
    for each added or deleted entry  $\langle fid, fname \rangle$  in  $r_1.ents$ 
      // Entry  $\langle fid, fname \rangle$  is potentially inconsistent. Fix up later.
      if  $fid \in \text{dom}(DISK)$  then
        IssueUupdate( $DISK(fid)$ ,  $\{ \}$ ) (15)

if  $r_2.ts > r_1.ts$  then (16)
  // The file's attributes are to be updated.
   $r_1.ts \leftarrow r_2.ts$ 
  if  $r_1.gpeers \neq r_2.gpeers$  then
     $r_1.gpeers \leftarrow r_2.gpeers$ 
    // When the replica's gold-peer set changes, I must reflect the
    // change to the parent dir entry.
    IssueUupdate( $r$ ,  $\{ \}$ )

  // Resolve potential conflicts on back pointers
  if  $r_1.bptrs \neq r_2.bptrs$  or  $r_1.deadBptr \neq r_2.deadBptr$  then
    IssueUupdate( $r_1$ ,  $r_1.bptrs \setminus r_2.bptrs$ ) (17)
     $r_1.bptrs \leftarrow r_2.bptrs$ 
     $r_1.deadBptr \leftarrow r_2.deadBptr$ 

  // If the last link to the replica is gone, erase the contents.
  if  $\neg IsLive(r_1)$  then
    if  $r_1$  is a regular file then
       $r_1.contents \leftarrow \text{None}$ 
    else
      for  $e \in r_1.ents \bullet e.valid$ 
        IssueUupdate( $DISK(e.fid)$ ,  $\{ \}$ ) (18)
       $r_1.ents \leftarrow \{ \}$ 

if Any of  $r_1$ 's attributes has changed then
  IssueCupdate( $r_1$ ) (19)

```

Fig. 14

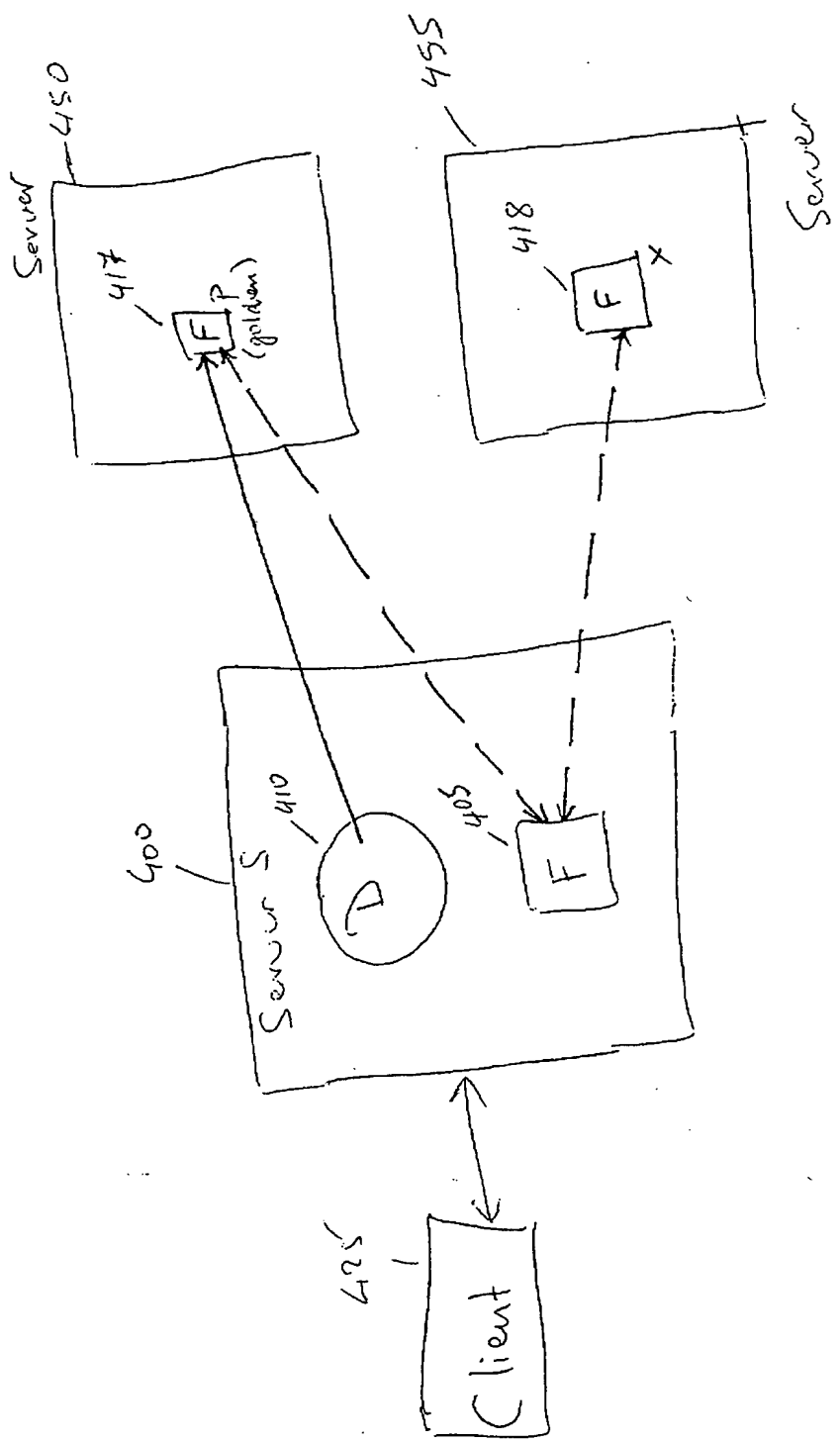


Figure 15

Protocol for adding a replica.

Constants:

M: Number of neighbors per replica.

MAXHOPS: The number of hops per a random walk (the usual value is 3)

#

# AddReplica is the main procedure that adds

# a replica of file F on the executing node.

#

AddReplica(F, G)

G: the set of gold replicas of F.

(G is obtained by looking up the parent directory)

g = Pick a random live node in G.

Send to g, "CreateReplica(F, myself)"

Wait for the contents to arrive.

Store contents and reply the client.

r = find the replica of F.

Send to g, "StartRandomWalk(F, myself)"

Wait for the set of neighbors N to arrive.

for n in N:

Add edge to n in r.

Send to n, "AddEdge(F, myself)"

SendReplicaContents(F, Sender):

F: the ID of the file

Sender: the node requesting replica creation.

r = find the replica of F

n = pick the replica closest to Sender among graphneighbors of r.

Send to n, "SendReplicaContents(F, Sender)"

SendReplicaContents(F, Sender):

F: the ID of the file

Sender: the node requesting replica creation.

r = find the replica of F

Send r to Sender.

StartRandomWalk(F, Sender):

F: the ID of the file

Sender: the node requesting replica creation.

Figure 16A



```

    r = find the replica of F
    N = {}
    for i = 0 to M-2:
        n = pick random graph neighbor in r.
        Send to n, "DoRandomWalk(F, 0, myself)"
        Receive nodeid from n.
        Add nodeid to N.
    Send N to Sender.

DoRandomWalk(F, hops, prevHopNode):
    F: the ID of the file
    hops: the number of hops made so far.

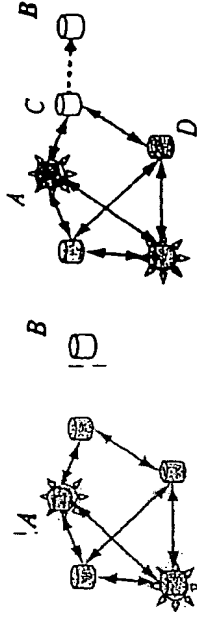
    if hops == MAXHOPS
        Send myself to prevHopNode
    else
        r = find the replica of F.
        n = pick random graph neighbor in r
        Send to n, "DoRandomWalk(F, hops + 1, myself)"
        Receive nodeid from n.
        Send nodeid to prevHopNode

AddEdge(F, peer):
    F: the ID of the file
    peer: the node to span edge to
    r = find the replica of F
    Add edge to peer in r

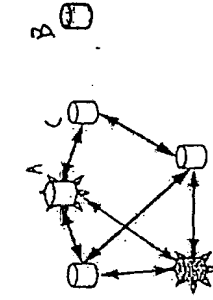
```

Figure 16B

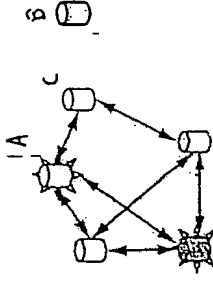
# Adding or deleting a bronze replica



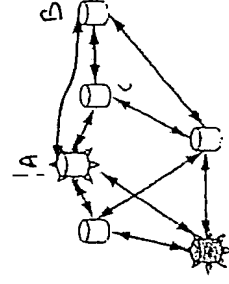
(1) B find gold replica A in the parent dir entry. B asks A to send contents.



(2) Find new peers using random walks starting from A.



(3) Issue RPCs from B to the picked nodes.



(4) Final state.

**proc IssueCupdate**

*r*: **Replica** // The replica of the file being updated.

---

$CLOG(r.fid) \leftarrow r.gpeers \cup r.peers$

**proc PropagateCupdate** // Runs periodically in the background

---

**for** ( $fid \mapsto targets$ )  $\in CLOG$

$r \leftarrow DISK(fid)$  // See <12>.

// Send the location of the parent dirs so that the target can  
// replicate them to ensure name-space containment.

$pDirs \leftarrow \{p: r.bptrs \bullet \langle p.fid, DISK(p.fid).gpeers \rangle\}$  // See <14>.

**for**  $n \in targets$

**send**  $\langle CUPDATE, r, pDirs \rangle$  **to**  $n$ .

**when receive**  $\langle CUPDATE-REPLY, ts \rangle$  from node  $n$

**if**  $CLOG(fid).ts = ts$

$CLOG(fid) \leftarrow CLOG(fid) \setminus \{n\}$

Remove  $fid$  from  $CLOG$  when  $CLOG(fid)$  becomes empty

---

**when receive**  $\langle CUPDATE, r, pDirs \rangle$

*r*: **Replica** // New replica contents.

$pDirs$ :  $\mathbb{P} \langle \text{FileID}, \mathbb{P} \text{NodeID} \rangle$  // Name and location of parent dirs.

---

**for**  $\langle pfid, ppeers \rangle \in pDirs$

CreateReplica( $pfid, ppeers$ )

ResurrectDirectory( $pfid$ )

UpdateReplica( $r$ )

**send**  $\langle CUPDATE-REPLY, r.ts \rangle$

**proc IssueUpdate**

*r*: **Replica** // The replica of the file

*del*:  $\mathbb{P} \langle \mathbf{FileID}, \mathbf{String} \rangle$  // Backpointers deleted from the replica.

---

**if** *r.fid*  $\in \text{dom}(\text{ULOG})$  **then**

*del*  $\leftarrow del \cup \text{ULOG}(r.fid)$

*ULOG*  $\leftarrow \text{ULOG} \oplus \{r.fid \mapsto del\}$

---

**proc ProcessUpdate** // Called periodically in the background.

---

**for** (*fid*  $\mapsto del$ )  $\in \text{ULOG}$

*r*  $\leftarrow \text{DISK}(fid)$  // See  $\langle 12 \rangle$ .

**for** *pfid*, *fname*  $\in del \cup r.bptrs$

        ResurrectDirectory(*pfid*)

*d*  $\leftarrow \text{DISK}(pfid)$

*valid* = *pfid*  $\in r.bptrs$  // Is this entry to be added?

*new* = ( $\langle fid, fname \rangle \mapsto \text{Dentry}(valid, r.ts, r.gpeers)$ )

*d.ents*  $\leftarrow (\langle fid, fname \rangle \triangleleft d.ents) \cup new$

**if** *d.ents* has changed

*d.ts*  $\leftarrow \text{Newtimestamp}()$

            IssueCupdate(*d*)

*ULOG*  $\leftarrow \{\}$

**proc** CreateReplica

*fid*: **FileID** // The ID of the file

*peers*:  $\mathbb{P}_1$  **NodeID** // The known set of gold peers of the file

**postconditions**:

*fid*  $\in$  dom(*DISK*)

---

**if** *fid*  $\in$  dom(*DISK*) **then**

**return**

**send**  $\langle \text{SEND-CONTENTS}, fid \rangle$  **to** random node *n*  $\in$  *peers*

    Wait until receive  $\langle \text{CONTENTS}, r, p\text{Dirs} \rangle$  from *n*

**for**  $\langle p\text{fid}, p\text{peers} \rangle \in p\text{Dirs}$

        CreateReplica(*pfid*, *ppeers*)

    UpdateReplica(*r*)

    Add edges between *r* and random existing replicas.

---

**when receive**  $\langle \text{SEND-CONTENTS}, fid \rangle$  from node *n*

---

*r*  $\leftarrow$  *DISK*(*fid*)

*pDirs*  $\leftarrow$   $\{p: r.\text{bptrs} \bullet \langle p.\text{fid}, \text{DISK}(p.\text{fid}).\text{gpeers} \rangle\}$  // See  $\langle 14 \rangle$ .

**send**  $\langle \text{CONTENTS}, r, p\text{Dirs} \rangle$  **to** *n*.

F.3. 20

**proc** ResurrectDirectory

*fid*: FileID

**preconditions:**

*fid*  $\in$  dom(*DISK*)

*fid* is a directory

**postconditions:**

IsLive(*DISK*(*fid*))

---

*r*  $\leftarrow$  *DISK*(*fid*)

**if** IsLive(*r*) **then**

**return**

ResurrectDirectory(*r*.deadBptr.pfid)

*r*.bptrs  $\leftarrow$  { *r*.deadBptr }

*r*.ts  $\leftarrow$  Newtimestamp()

IssueCupdate(*r*)

**let**  $\langle$ pfid, fname $\rangle =$  *r*.deadBptr •

*d*  $\leftarrow$  *DISK*(pfid)

*d*.ents( $\langle$ fid, fname $\rangle$ )  $\leftarrow$  **Dentry**(true, *r*.ts, *r*.gpeers) <sup>(20)</sup>

*d*.ts  $\leftarrow$  Newtimestamp()

    IssueCupdate(*d*)

A	$fid /$ $fid_{alice}$	5:A 12:A	ents= $\{\langle fid_{alice}, \text{"alice"}, 12:A \rangle\}$ ents= $\{*\langle fid_{bar}, \text{"bar"}, 13:B \rangle\}$
---	--------------------------	-------------	---

Fig. 22

A	<i>fid</i> <sub>/</sub>	5:A	ents={ $\langle$ <i>fid</i> <sub>foo</sub> , "foo",... $\rangle$ }
	<i>fid</i> <sub>alice</sub>	6:A	ents={}
	<i>fid</i> <sub>foo</sub>	8:A	bptrs= $\langle$ <i>fid</i> <sub>/</sub> , "foo" $\rangle$
B	<i>fid</i> <sub>/</sub>	5:A	ents={ $\langle$ <i>fid</i> <sub>foo</sub> , "foo",... $\rangle$ }
	<i>fid</i> <sub>bob</sub>	7:B	ents={}
	<i>fid</i> <sub>foo</sub>	8:A	bptrs= $\langle$ <i>fid</i> <sub>/</sub> , "foo" $\rangle$



A	$fid_{/}$	10:A	$ents = \{ * \langle fid_{foo}, "foo", ... \rangle \}$
	$fid_{alice}$	11:A	$ents = \{ \langle fid_{foo}, "foo1", 12:A \rangle \}$
	$fid_{foo}$	12:A	$bptrs = \langle fid_{alice}, "foo1" \rangle$
B	$fid_{/}$	5:A	$ents = \{ \langle fid_{foo}, "foo", ... \rangle \}$
	$fid_{bob}$	7:B	$ents = \{ \}$
	$fid_{foo}$	8:A	$bptrs = \langle fid_{/}, "foo" \rangle$

A	<i>fid</i> /	10:A	ents= $\{*\langle \mathbf{fid}_{foo}, \text{"foo"}, \dots \rangle\}$
	<i>fid</i> <sub>alice</sub>	11:A	ents= $\{\langle \mathbf{fid}_{foo}, \text{"foo1"}, 12:A \rangle\}$
	<i>fid</i> <sub>foo</sub>	12:A	bptrs= $\langle \mathbf{fid}_{alice}, \text{"foo1"} \rangle$
B	<i>fid</i> /	9:B	ents= $\{*\langle \mathbf{fid}_{foo}, \text{"foo"}, \dots \rangle\}$
	<i>fid</i> <sub>bob</sub>	11:B	ents= $\{\langle \mathbf{fid}_{foo}, \text{"foo2"}, 12:B \rangle\}$
	<i>fid</i> <sub>foo</sub>	12:B	bptrs= $\langle \mathbf{fid}_{bob}, \text{"foo2"} \rangle$

Fig. 25

A	$fid_{/}$	10:A	$ents = \{ * \langle fid_{foo}, "foo", ... \rangle \}$
	$fid_{alice}$	14:A	$ents = \{ * \langle fid_{foo}, "foo1", 12:B \rangle \}$
	$fid_{bob}$	11:B	$ents = \{ \langle fid_{foo}, "foo2", 12:B \rangle \}$
	$fid_{foo}$	12:B	$bptrs = \langle fid_{bob}, "foo2" \rangle$
B	$fid_{/}$	9:B	$ents = \{ * \langle fid_{foo}, "foo", ... \rangle \}$
	$fid_{bob}$	11:B	$ents = \{ \langle fid_{foo}, "foo2", 12:B \rangle \}$
	$fid_{foo}$	12:B	$bptrs = \langle fid_{bob}, "foo2" \rangle$

Fig. 26

A	$fid_{/}$	10:A	$ents = \{ * \langle fid_{foo}, "foo", ... \rangle \}$
	$fid_{alice}$	14:A	$ents = \{ * \langle fid_{foo}, "foo1", 12:B \rangle \}$
	$fid_{bob}$	11:B	$ents = \{ \langle fid_{foo}, "foo2", 12:B \rangle \}$
	$fid_{foo}$	12:B	$bptrs = \langle fid_{bob}, "foo2" \rangle$
B	$fid_{/}$	10:A	$ents = \{ * \langle fid_{foo}, "foo", ... \rangle \}$
	$fid_{bob}$	11:B	$ents = \{ \langle fid_{foo}, "foo2", 12:B \rangle \}$
	$fid_{foo}$	12:B	$bptrs = \langle fid_{bob}, "foo2" \rangle$

F.5. 27

A	$fid_{/}$ $fid_{foo}$	5:A 6:A	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$ bptrs= $\{\langle fid_{foo}, "foo" \rangle\}$
B	$fid_{/}$ $fid_{foo}$	5:A 6:A	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$ bptrs= $\{\langle fid_{foo}, "foo" \rangle\}$

Fig. 28

A	$fid_{/}$ $fid_{foo}$	10:A 11:A	ents= $\{*\langle fid_{foo}, \text{"foo"}, 11:A \rangle\}$ bptrs= $\{\}$
B	$fid_{/}$ $fid_{foo}$	5:A 6:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 6:A \rangle\}$ bptrs= $\{\langle fid_{foo}, \text{"foo"} \rangle\}$

Fig. 29

A	$fid /$ $fid_{foo}$	10:A 11:A	ents= $\{*\langle fid_{foo}, \text{"foo"}, 11:A \rangle\}$ bptrs= $\{\}$
B	$fid /$ $fid_{foo}$	5:A 11:B	ents= $\{\langle fid_{foo}, \text{"foo"}, 6:A \rangle\}$ bptrs= $\{\langle fid_{foo}, \text{"foo"} \rangle\}$

Fig. 30

A	$\textit{fid}_/$ $\textit{fid}_{foo}$	12:A 11:B	ents= $\{\langle \textit{fid}_{foo}, \text{“foo”}, 11:B \rangle\}$ bptrs= $\{\langle \textit{fid}_{foo}, \text{“foo”} \rangle\}$
B	$\textit{fid}_/$ $\textit{fid}_{foo}$	12:A 11:B	ents= $\{\langle \textit{fid}_{foo}, \text{“foo”}, 11:B \rangle\}$ bptrs= $\{\langle \textit{fid}_{foo}, \text{“foo”} \rangle\}$

Fig. 31



A	$fid_{/}$ $fid_{foo}$	12:B 11:A	ents= $\{*\langle fid_{foo}, \text{"foo"}, 11:A \rangle\}$ bptrs= $\{\}$
B	$fid_{/}$ $fid_{foo}$	12:B 11:A	ents= $\{*\langle fid_{foo}, \text{"foo"}, 11:A \rangle\}$ bptrs= $\{\}$

Fig. 32

A	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 6:A \rangle\}$
	$fid_{foo}$	6:A	ents= $\{\}$ , bptrs= $\{\langle fid_{/}, \text{"foo"} \rangle\}$
B	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 6:A \rangle\}$
	$fid_{foo}$	6:A	ents= $\{\}$ , bptrs= $\{\langle fid_{/}, \text{"foo"} \rangle\}$

Fig. 33

A	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 6:A \rangle\}$
	$fid_{foo}$	10:A	ents= $\{\langle fid_{bar}, \text{"bar"}, 11:A \rangle\}$ , bptrs= $\{\langle fid_{/}, \text{"foo"} \rangle\}$
	$fid_{bar}$	11:A	bptrs= $\{\langle fid_{foo}, \text{"bar"} \rangle\}$
B	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 6:A \rangle\}$
	$fid_{foo}$	6:A	ents= $\{\}$ , bptrs= $\{\langle fid_{/}, \text{"foo"} \rangle\}$

Fig. 34

A	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 6:A \rangle\}$
	$fid_{foo}$	10:A	ents= $\{\langle fid_{bar}, \text{"bar"}, 11:A \rangle\}$ , bptrs= $\{\langle fid_{/}, \text{"foo"} \rangle\}$
	$fid_{bar}$	11:A	bptrs= $\{\langle fid_{foo}, \text{"bar"} \rangle\}$
B	$fid_{/}$	8:B	ents= $\{*\langle fid_{foo}, \text{"foo"}, 10:B \rangle\}$
	$fid_{foo}$	10:B	ents= $\{\}$ , bptrs= $\{\}$

Fig. 35

A	<i>fid</i> /	8:B	ents={ * $\langle$ <i>fid</i> <sub>dir</sub> , “dir”, 10:B $\rangle$ }
	<i>fid</i> <sub>dir</sub>	10:B	ents={}, bptrs={}
	<i>fid</i> <sub>foo</sub>	11:A	bptrs={ $\langle$ <i>fid</i> <sub>dir</sub> , “foo” $\rangle$ }
B	<i>fid</i> /	8:B	ents={ * $\langle$ <i>fid</i> <sub>dir</sub> , “dir”, 10:B $\rangle$ }
	<i>fid</i> <sub>dir</sub>	10:B	ents={}, bptrs={}

Fig. 36

A	<i>fid</i> /	12:A	ents={ $\langle \textit{fid}_{dir}, \text{"dir"}, 13:A \rangle$ }
	<i>fid</i> <sub>dir</sub>	13:A	ents={ $\langle \textit{fid}_{foo}, \text{"foo"}, 11:A \rangle$ }
	<i>fid</i> <sub>foo</sub>	11:A	bptrs={ $\langle \textit{fid}_{dir}, \text{"foo"} \rangle$ }
B	<i>fid</i> /	12:A	ents={ $\langle \textit{fid}_{dir}, \text{"dir"}, 13:A \rangle$ }
	<i>fid</i> <sub>dir</sub>	13:A	ents={ $\langle \textit{fid}_{foo}, \text{"foo"}, 11:A \rangle$ }

Fig. 37

A	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, \text{"dir"}, 10:B\rangle\}$
	$fid_{dir}$	10:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 11:A\rangle\}$ , bptrs= $\{\langle fid_{/}, \text{"dir"}\rangle\}$
	$fid_{foo}$	11:A	bptrs= $\{\langle fid_{dir}, \text{"foo"}\rangle\}$
B	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, \text{"dir"}, 10:B\rangle\}$
	$fid_{dir}$	10:B	ents= $\{\}$ , bptrs= $\{\}$

Fig. 38

A	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, "dir", 10:B\rangle\}$
	$fid_{dir}$	10:A	ents= $\{\langle fid_{foo}, "foo", 11:A\rangle\}$ , bptrs= $\{\langle fid_{/}, "dir"\rangle\}$
	$fid_{foo}$	11:A	bptrs= $\{\langle fid_{dir}, "foo"\rangle\}$
B	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, "dir", 10:B\rangle\}$
	$fid_{dir}$	10:A	ents= $\{\langle fid_{foo}, "foo", 11:A\rangle\}$ , bptrs= $\{\langle fid_{/}, "dir"\rangle\}$

Fig. 39



A	$fid_{/}$	13:A	ents= $\{\langle fid_{dir}, \text{"dir"}, 10:A \rangle\}$
	$fid_{dir}$	10:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 11:A \rangle\}$ , bptrs= $\{\langle fid_{/}, \text{"dir"} \rangle\}$
	$fid_{foo}$	11:A	bptrs= $\{\langle fid_{dir}, \text{"foo"} \rangle\}$
B	$fid_{/}$	13:A	ents= $\{\langle fid_{dir}, \text{"dir"}, 10:A \rangle\}$
	$fid_{dir}$	10:A	ents= $\{\langle fid_{foo}, \text{"foo"}, 11:A \rangle\}$ , bptrs= $\{\langle fid_{/}, \text{"dir"} \rangle\}$

Fig. 40

```

// Called every third night for every replica on the node.
proc GarbageCollection
  LiveNodes:  $\mathbb{P}_1$  NodeID // Set of live nodes.
  r: Replica // Replica to be inspected.
  EXPIRE: integer // Dead-replica expiration period, e.g., a month.

// Remove old tombstones. Removing after EXPIRE seconds is safe
// because we cannot receive any new update with timestamp older
// than EXPIRE after removing r.
if  $\neg \text{IsLive}(r)$  and  $r.ts < \text{Newtimestamp} - EXPIRE$  then
  DISK  $\leftarrow \{r.fid\} \triangleleft DISK$ 
  return

r'  $\leftarrow \text{Deepcopy}(r)$ 

// Remove dead entries in the directory
if r' is a directory then
  for (key  $\mapsto$  val): r'.ents •
    if  $\neg val.valid$  and  $val.ts < \text{Newtimestamp}() - EXPIRE$  then
      r'.ents  $\leftarrow \{key\} \triangleleft r'.ents$ 
  if at least one entry has been removed from r'.ents then
    r'.ts  $\leftarrow \text{NewTimestamp}()$ 
    UpdateReplica(r)

// If some gold peers are found dead, recreate one elsewhere
if  $me \in r.gpeers$  and  $r.gpeers \not\subseteq \text{Livenodes}$  then
  newNodes  $\leftarrow \text{Pick } ||r.gpeers \setminus \text{Livenodes}||$  random live nodes.
  r'.gpeers  $\leftarrow r.gpeers \setminus \text{Livenodes} \cup newNodes$ 
  r'.ts  $\leftarrow \text{NewTimestamp}()$ 
  UpdateReplica(r)

// If we find graph edges to dead nodes, re-span it.
for  $n \in r.peers \setminus \text{Livenodes}$ 
  Add edges between r and a random replica.
  r.peers  $\leftarrow r.peers \cap \text{Livenodes}$ 

```

$\forall f: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(f) \bullet$   
(  $\langle d.\text{fid}, \text{fname} \rangle \in f.\text{bptrs}$   
 $\Rightarrow d \in \text{dom}(\text{DISK}) \text{ and } \text{IsLive}(d)$  )

Fig. 42

1

$\forall d: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(d) \bullet$   
 $( f: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(f) \bullet$   
 $\langle d.\text{fid}, \text{fname} \rangle \mapsto \text{ent} \in d.\text{ents} \text{ and } \text{ent.valid}$   
 $\text{and } \langle d.\text{fid}, \text{fname} \rangle \notin f.\text{bptr}$   
 $\Rightarrow \Diamond \langle f.\text{fid}, \text{fname} \rangle \mapsto \text{ent} \notin d.\text{ents} )$

Fig. 43

$\forall d: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(d) \bullet$   
 $( \langle f.\text{fid}, \text{fname} \rangle \mapsto \text{ent} ) \in d.\text{ents} \text{ and } \text{ent}.\text{valid}$   
 $\Rightarrow f: \text{ran}(\text{DISK}') \text{ and } \text{IsLive}(f) )$

Fig. 44

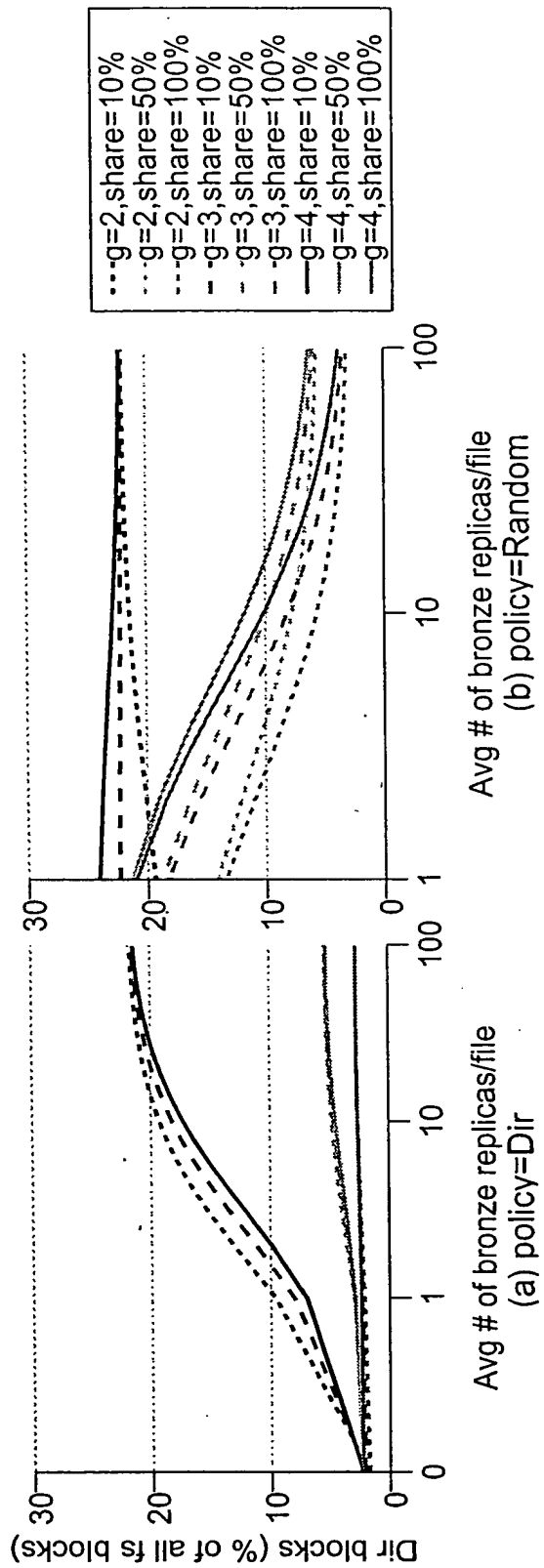


Fig. 45